ULTRA-LOW-POWER DOMAIN-SPECIFIC MULTIMEDIA PROCESSORS

Arthur Abnous and Jan Rabaey
Dept. of Electrical Engineering and Computer Sciences
University of California, Berkeley

Abstract - Programmability is an important requirement for portable computing and communication devices that must be flexible enough to accommodate a variety of multimedia services and communication capabilities. However, compared to dedicated, application-specific solutions, programmable devices often incur significant performance and power penalties. We present a hybrid architecture template that can be used to implement ultra-low-power programmable processors for signal processing applications.

INTRODUCTION

An important trend driving the electronics industry is the growing demand for increasingly sophisticated portable computing and communication devices. A major component of the functionality of these devices consists of a variety of digital signal processing capabilities that are required to communicate and process a variety of multimedia data. The increasing levels of integration and performance that are required for future single-chip, full-function multimedia processing devices will be accompanied by increasing levels of power dissipation [1]. Without significant advances in energy-efficient design techniques and architectures, battery life and system cost constraints will limit the capabilities of portable multimedia devices.

An important requirement for advanced portable computing and communication devices is programmability. These devices will have to be flexible enough to handle a variety of multimedia services and standards (e.g., different speech coding or video decompression schemes) and to provide a variety of communication capabilities (e.g., cellular telephony and wireless networking). The design of these devices should, therefore, be based on programmable and reconfigurable components.

The drawback of programmable devices is that they incur significant performance and power penalties. In this paper, we present a hybrid architecture template that can be used to implement ultra-low-power programmable processors for digital signal processing applications.

ARCHITECTURAL STRATEGY

The difficulty in achieving high levels of energy-efficiency in a programmable processor stems from the inherent trade-off between flexibility and energy-efficiency (Figure 1). Programmability requires *generalized* computation, storage, and communication structures, which can be used to implement different kinds of algorithms.

0-7803-3134-6/96 $5.00 © 1996 IEEE

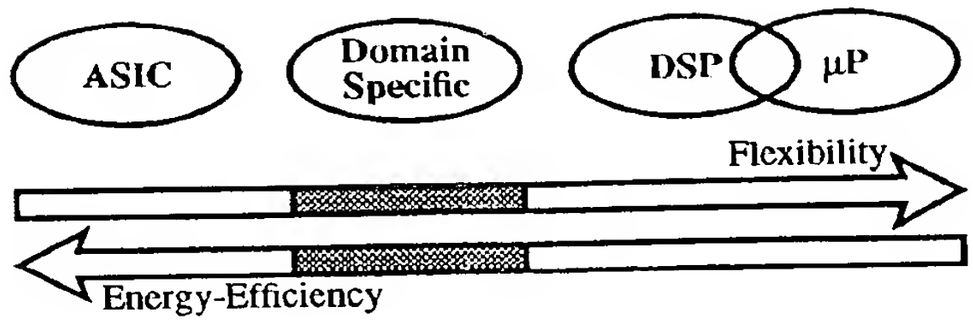


Figure 1: Flexibility vs. Energy-Efficiency

Efficiency, on the other hand, dictates the use of dedicated structures that can fully exploit the properties of a given algorithm. While conventional programmable architectures (e.g., microprocessors and programmable digital signal processors) can be programmed to perform virtually any computational task, they achieve this flexibility by incurring the significant overhead of fetching, decoding, and executing a stream of instructions on complex, general-purpose datapaths. For example, the processing core of the TMS320C5x family of general-purpose DSPs from Texas Instruments draws a total current of 55 mA from a 5V supply when executing a typical DSP application program with a 20% mix of multiply-accumulate operations [2]. Instruction fetching and decoding is responsible for 42 mA, i.e., 76%, of the total supply current. Clearly, a heavy penalty is paid for performing computations on a general-purpose datapath under the control of an instruction stream.

The flexibility of general-purpose processors is highly desirable for handling general computing tasks. Signal processing algorithms, on the other hand, have intrinsic properties that makes them more amenable to efficient implementations that do not require the full flexibility of a general-purpose device. These algorithms exhibit high degrees of parallelism and are dominated by a few regular kernels of computation that are responsible for a large fraction of execution time and energy. While an application-specific implementation can always achieve the highest level of energy-efficiency, significant energy savings can be achieved by executing the dominant computational kernels of a given class of signal processing applications with common features on dedicated, optimized processing elements that can execute those kernels with minimum energy overhead. This leads to the concept of *domain-specific* processors, which are optimized for a given subset, or domain, of algorithms. This optimization involves trading off the flexibility of a general-purpose programmable device to achieve higher levels of energy-efficiency, while maintaining the flexibility to handle a variety of algorithms within the domain of interest.

VSELP Case Study

To evaluate the potential impact of executing the dominant computational kernels of signal processing algorithms on dedicated, optimized processing elements, we have done an analysis of the VSELP Speech coder [3]. The goal of the analysis was to evaluate the computational complexity of the algorithm and to identify the dominant kernels of the algorithm that are responsible for a large fraction of the execution time and energy of the algorithm. This analysis also establishes a lower

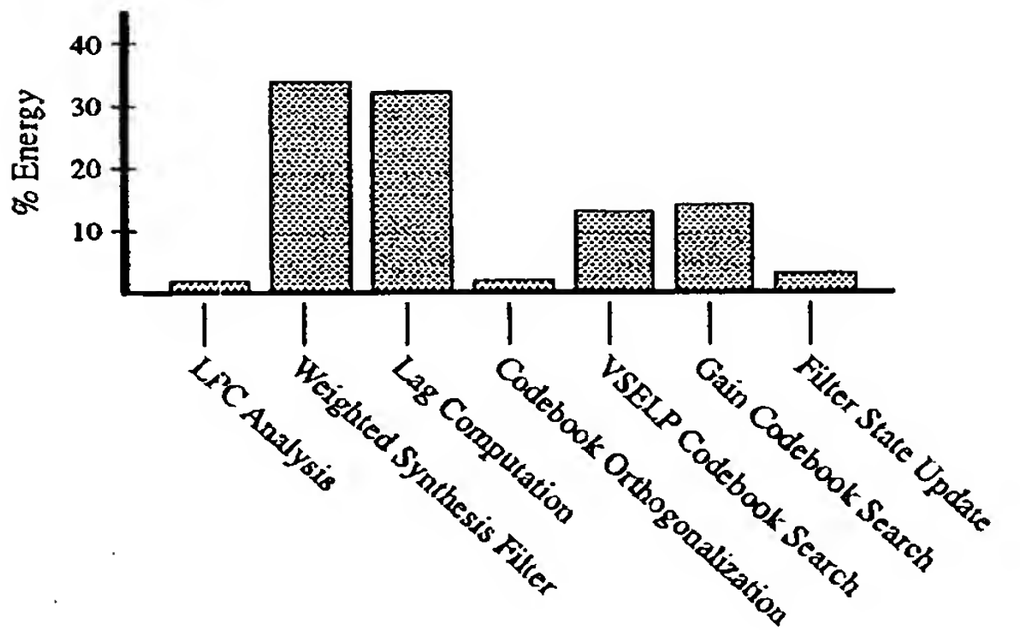


Figure 2: Arithmetic Energy Profile for the VSELP Speech Coder

bound for energy consumption. First, the dynamic execution profile of the algorithm is extracted. This information is then post-processed to produce a dynamic operation count, which is then fed into a high-level power analysis tool [4]. The result of this analysis is a power dissipation profile of the algorithm. Figure 2 shows the energy profile for the arithmetic operations of the VSELP speech coder for actual voice data. It can be seen that 92% of the total energy is consumed in the inner loops of four functions: weighted synthesis filter, lag computation, VSELP codebook search, and gain codebook search. Vector dot-product calculation accounts for 65% of total power in VSELP, so the ability to compute dot products at the lowest possible supply voltage with a minimum of energy overhead can have a large impact on the overall energy consumption of this algorithm.

Energy-Efficient Design Techniques

Previous research in the area of low-power design has led to a number of important architectural techniques that have been applied successfully to implement application-specific devices for signal processing applications [5].

One of the most effective ways of reducing the energy consumption of a circuit is to reduce the supply voltage because the energy of a switching event drops quadratically with the supply voltage. This drop in energy consumption is accompanied by reduced circuit speeds, so it is necessary to compensate for this degradation in circuit speed by exploiting concurrency. Thus, an important design objective for an energy-efficient architecture is the ability to execute concurrent threads of computation at low supply voltages.

Other important techniques to reduce power dissipation are aimed at minimizing switching capacitance. This is achieved by (a) exploiting locality of reference with distributed computational structures, minimizing global interactions, (b) enforcing a demand-driven policy that eliminates switching activity in unused modules (often called power management), and (c) preserving temporal correlations in data streams

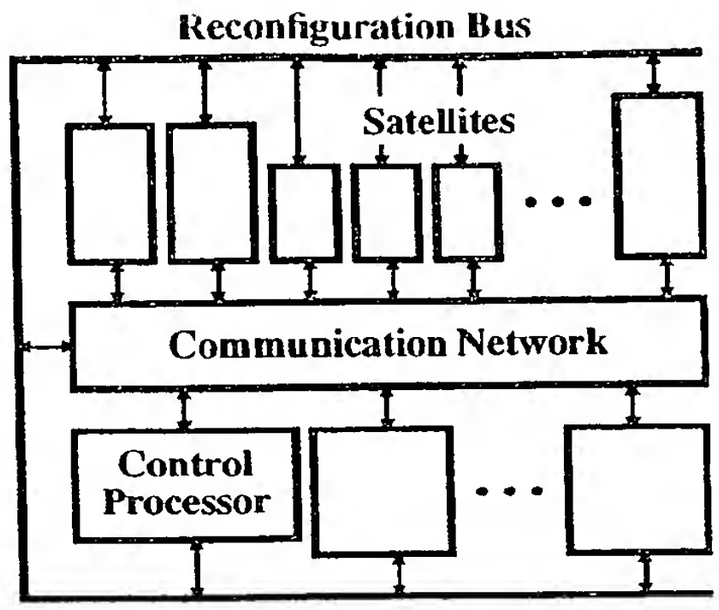


Figure 3: Architecture Template

by minimizing the degree of hardware sharing.

Most of these techniques can only be applied with limited success to conventional programmable architectures that are based on a complex datapath used for executing a relatively large instruction set under the supervision of a global instruction fetch and decode controller. An energy-efficient architecture must be able to exploit these techniques effectively and combine them into a unified architectural framework. This principle was the fundamental objective for the architecture presented in this paper.

THE PROPOSED ARCHITECTURE

The architecture that we are proposing is based on the reusable *template* illustrated in Figure 3. This template can be used to implement a domain-specific *instance*, which can then be programmed to implement a variety of algorithms within the given domain of interest. All instances of the architecture template share a fixed set of control and communication primitives. The type and number of processing elements in a given domain-specific instance, however, can vary; they depend on the properties of a particular domain of interest.

The architecture template consists of a *control processor* (a general-purpose microprocessor core), surrounded by a heterogeneous array of autonomous, special-purpose *satellite processors*. The computational demand on the control processor is minimal, as its main task is to configure the satellite processors and the communication network (over the reconfiguration bus) and to manage the overall control flow of a given signal processing algorithm. The dominant, energy-intensive computational kernels of a given domain of algorithms are implemented as a set of independent, concurrent threads of computation on the satellite processors. All processors in the system communicate over a reconfigurable communication network. Computation and communication activities are coordinated via a distributed data-driven control mechanism.

A given application implemented on a domain-specific processor consists of a set

of concurrent processes that run on the various hardware resources of the processor and are managed by the control processor. Some of these processes correspond to the dominant kernels of the given application program and run on satellite processors under the supervision of the control processor. Other processes run on the control processor under the supervision of a simple interrupt-driven foreground/background system for relatively simple applications or under the supervision of a real-time kernel for more complex applications. The control processor uses the available satellite processors and the reconfigurable interconnect to compose the dataflow graph corresponding to a given kernel of computation in hardware. The nodes of the dataflow graph correspond to the satellite processors, and the arcs correspond to links in the reconfigurable interconnect. Each arc in the dataflow graph is assigned a dedicated link in the communication network. This ensures that all temporal correlations in a given stream of data are preserved and the amount of switching activity is minimized.

Algorithms in a given domain of applications, e.g., CELP-based speech coders [6], share a common set of operations, e.g., LPC analysis, synthesis filter, codebook search. When and how these operations are applied depends on the particular algorithm and is managed by the control processor. The underlying details and parameters of the various tasks in a given domain vary (from algorithm to algorithm), and they are supported by the reconfigurability of the satellite processors and the communication network.

The architecture that we are proposing has the benefit of reusability because (a) there is a set of predefined control and communication primitives that are fixed across all domain-specific instances of the template, and (b) predefined satellite processors can be placed in a library and reused in the design of different processors.

Satellite Processors

The computational core of the architecture consists of a heterogeneous array of autonomous, special-purpose satellite processors. These processors are optimized to execute specific tasks efficiently, with minimal energy overhead. Instead of executing all computations in a general-purpose datapath, as is commonly done in conventional programmable processors, the energy-intensive kernels of an algorithm are executed on optimized satellite processors without the overhead of fetching and decoding an instruction stream. Data tokens are processed by a cluster of interconnected satellite processors in a highly pipelined manner; each satellite processor forms one stage of a pipeline. In addition, each satellite processor can be further pipelined internally. Additionally, multiple pipelines corresponding to multiple independent kernels can be executed in parallel. These capabilities allow efficient processing at very low supply voltages. Additionally, for bursty applications with dynamically varying throughput requirements, dynamic scaling of the supply voltage is used to meet the throughput requirements of the algorithm at the minimum supply voltage.

As mentioned earlier, satellite processors perform specific tasks. Some examples include address generators, memory modules, multiply-accumulate (MAC) processors for computing vector dot products (Figure 4), and add-compare-select (ACS) processors for Viterbi decoding. The exact bevavior of a satellite processor is con-

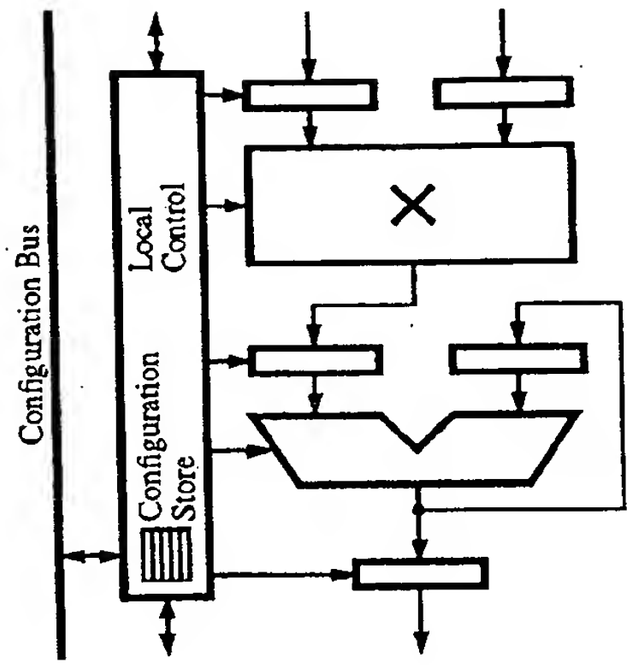


Figure 4: Multiply-Accumulate Processor

trolled by a set of parameters. While most satellite processors are fairly dedicated, others might support a higher degree of flexibility to allow the implementation of a wider range of kernels. Some satellite processors are, therefore, implemented as programmable datapaths which can be configured at the level of arithmetic operators.

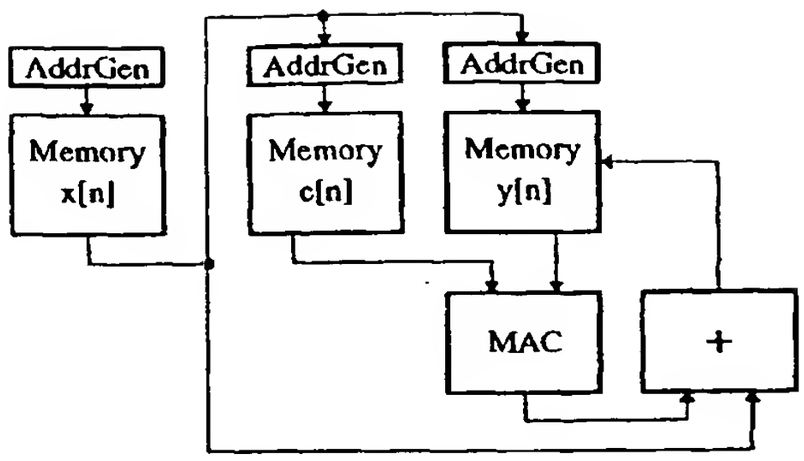
The behavior of a satellite processor is specified by a few parameters or simple instructions that are stored in a local configuration store. The configuration store consists of a small number of registers and can, thus, be configured quickly by the control processor via a wide, global reconfiguration bus. Each satellite processor has two sets of configuration registers. This allows the control processor to configure one register set for an upcoming kernel, while the satellite processor is executing another kernel as specified by the alternate register set. When the current task is completed, the next task can be started immediately.

Figure 5 illustrates how one of the energy-intensive functions of the VSELP speech coder, the weighted synthesis filter, is mapped onto satellite processors.

Communication Network

The communication network is reconfigured by the control processor to implement the arcs of the dataflow graph corresponding to the dominant kernel being implemented as a hardware process. As mentioned earlier, each arc in the dataflow graph is assigned a dedicated link in the communication network. This ensures that all temporal correlations in a given stream of data are preserved, and the amount of switching activity is minimized.

To minimize the energy consumption of the communication network, a number of techniques are employed. These include (a) reduced voltage swings, (b) clustered satellite processors with local networks, and (c) segmentation of buses into smaller, less capacitive sections by break switches.



$$y[1 \leq n \leq N] = x[n] + \sum_{i=1}^{N_P} c_i \cdot y[n-i]$$

Figure 5: The VSELP Synthesis Filter Mapped onto Satellite Processors

Distributed Data-Driven Control

As mentioned earlier, a given process running on the satellite processors implements the dataflow graph of the corresponding kernel directly on hardware units. This suggests a data-driven scheme for coordinating computation and communication activities across the system [7], whereby all activities are synchronized by passing data tokens. This scheme is distributed, and compared to a traditional centralized control scheme, it is highly modular and scalable and provides for the graceful synchronization of a large number of processing elements.

A data-driven control mechanism, however, has additional benefits from the point of view of reducing energy consumption. By distributing control across small local controllers, the physical capacitance associated with a large, centralized controller is avoided. Additionally, a data-driven mechanism provides a clear and elegant framework for implementing a demand-driven policy for managing switching activity for all hardware blocks. Execution of a hardware module is triggered by the arrival of tokens. When there are no tokens to be processed at a given module, no switching activity occurs in that module.

Synchronization and Clocking

While the distributed data-driven control mechanism described in the previous section enables important energy savings, it requires a proper handshaking mechanism. The issue is further complicated by the fact that individual satellite processors might have different and varying supply voltages and data rates. Also, the ability to freely select and interconnect satellite processors from a pre-existing library without having to redesign for a new set of timing constraints is a highly desirable feature. All of these suggest an asynchronous timing scheme, at least at the global level. An asynchronous scheme provides a high degree of flexibility and scalability, while

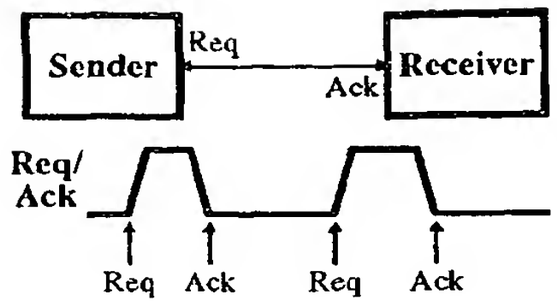


Figure 6: Single-Wire, Two-Phase Asynchronous Handshaking Protocol

avoiding the energy overhead of distributing and managing a global clock. This overhead can be as high as 28% of total power dissipation for some high-performance designs [8].

Asynchronous communication requires a proper handshaking protocol. An efficient signalling scheme is shown in Figure 6. A single wire is used alternately by the sender and receiver of a data token. This scheme combines the efficiency of a conventional two-phase protocol (high throughput and minimum number of transitions) with the desirable return-to-zero characteristic of the conventional four-phase protocol [9].

For a given processor, a fully asynchronous implementation might not be the most efficient as it requires the overhead of generating completion signals. Thus, it might be more efficient to use traditional synchronous design methodologies for implementing individual satellite processors. Therefore, a locally synchronous, globally asynchronous timing methodology, as shown in Figure 7, is used. In this scheme, a local clock is generated under the control of incoming data tokens. Arrival of a data tokens restarts the clock generator, and when the processor is done processing the data token, the clock generator is shut down. The period of the clock generator is programmable and tracks the local logic delays. In the particular arrangement shown in Figure 7, synchronization failures due to metastability are not possible, and the processor can operate reliably. If the logic block is pipelined however, and can accept additional data tokens after the clock generator has started running, then the potential for metastability exists. Value-safe operation can be ensured by detecting metastable conditions and stretching the period of the local clock signal accordingly [10].

PROJECTED RESULTS

Current estimates indicate that compared to general-purpose programmable signal processors, the proposed reconfigurable multiprocessor architecture can reduce the energy consumption of CELP-based speech coding algorithms by up to an order of magnitude, while maintaining a high level of programmability that can be used to implement various types of speech coding algorithms. Currently, the most energy-efficient signal processor for speech coding algorithms is the processor from Mitsubishi [11] which dissipates 36 mW ($V_{dd} = 1.8$ V, 0.5-μm CMOS technology) for the Japanese half-rate speech coding standard, which requires 23.4 MOPS on that processor (VSELP requires 7.8 MOPS). We project that the VSELP speech coder

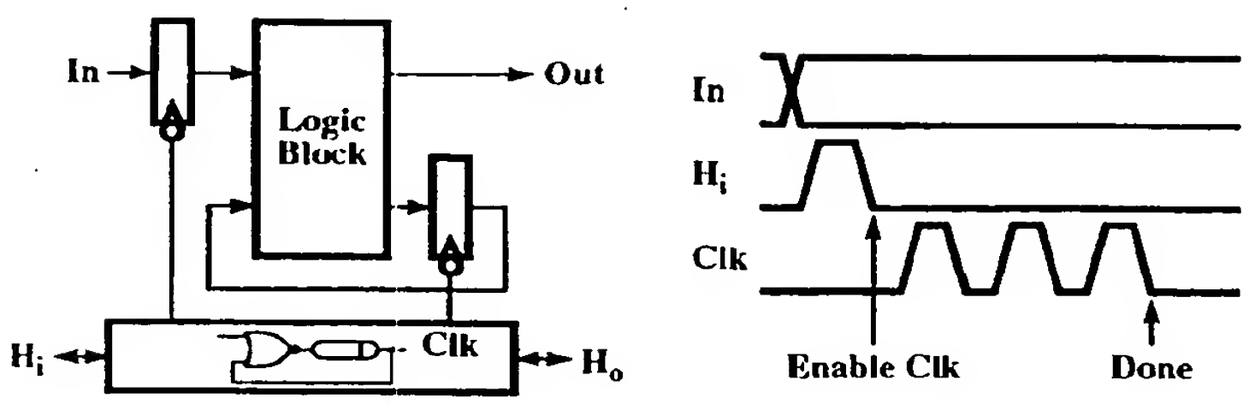


Figure 7: Locally Synchronous, Globally Asynchronous Timing Scheme

can be implemented on our architecture in a conservative 0.6-μm CMOS technology in under 5 mW.

SUMMARY

A programmable, heterogeneous multiprocessor architecture that can be used to implement ultra-low-power domain-specific signal processors has been presented. The architecture exploits the high degrees of concurrency and regularity that are present in many signal processing kernels.

Future work is focused on further evaluation of the power-saving capabilities of the architecture and a methodology for mapping a signal processing domain onto the architecture template. A prototype chip aimed at ultra-low-power implementations of speech coding algorithms is being implemented.

ACKNOWLEDGMENTS

We would like to thank Marlene Wan for her assistance with the VSELP case study. We would also like to thank Lisa Guerra and David Lidsky for their suggestions on improving this manuscript.

REFERENCES

[1] H. Sasaki, "Multimedia Complex on a Chip," *ISSCC Digest of Technical Papers*, pp. 16-19, February 1996.

[2] J. Bradley, *Calculation of TMS320C5x Power Dissipation*, Technical Application Report SPRA030, Texas Instruments, 1993.

[3] I. Gerson and M. Jasiuk, "Vector Sum Excited Linear Prediction (VSELP) Speech Coding at 8Kbps," *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, pp. 461-464, April 1990.

[4] D. Lidsky and J. Rabaey, "Early Power Exploration - A World Wide Web Applicaion," *Proceedings of the 33rd Design Automation Conference*, pp. 27-32, June 1996.

[5] A. P. Chandrakasan and R. W. Brodersen, *Low Power Digital CMOS Design*, Kluwer Academic Publishers, Norwell, Massachusetts, 1995.

[6] M. R. Schroeder and B. S. Atal, "Code-Excited Linear Prediction (CELP): High Quality Speech at Very Low Bit Rates," *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, pp. 937-940, 1985.

[7] A. Yeung and J. Rabaey, "A 2.4GOPS Data-Driven Reconfigurable Multiprocessor IC for DSP," *ISSCC Digest of Technical Papers*, pp. 108-109, February 1995.

[8] D. Dobberpuhl *et al.*, "A 200MHz Dual-Issue CMOS Microprocessor," *ISSCC Digest of Technical Papers*, pp. 106-107, February 1992.

[9] C. Mead and L. Conway, *Introduction to VLSI Systems*, Chapter 7, Addison-Wesley, Reading, MA, 1980.

[10] M. Pěchouček, "Anomalous Response Times of Input Synchronizers," *IEEE Transactions on Computers*, pp. 133-139, February 1976.

[11] T. Shiraishi, "A 1.8V 36mW DSP for the Half-Rate Speech CODEC," *Proceedings of the IEEE Custom Integrated Circuits conference*, pp. 371-374, May 1996.